QALS — Implementation Plan

Six phases from fork to venue

QALS project · qalarc network · 09 September 2026
generated from implementation_plan.md · not legal or financial advice

QALS — Implementation Plan & Engineering Roadmap

Project: Qalnet (private L1 forked from iotaledger/iota) → QALS credit, compute, DeFi, identity
Date: 2026-09-06 · **Status:** v1 planning baseline **Sources:** `03_qals_architecture/QALS_ARCHITECTURE.md` (§6 build plan), `QALS_BACKING_DESIGN.md`, `05_defi_trading/QALX_DEFI_PLATFORM.md` + `DEFI_PLAN.md`, `07_compute_marketplace/COMPUTE_MARKETPLACE.md`, `01_iota_deep_research/IOTA_COMPONENT_MAP.md` (tooling entry points), `06_bank_exchange/BANK_EXCHANGE_PLAN.md` (compliance track)

---

1. Principles

These principles govern every phase. They exist to keep a 104-crate upstream codebase mergeable, auditable, and shippable by a small team + AI agents.

1. **Upstream-first forking.** The `qalnet` repo tracks `iotaledger/iota` as `upstream`. Every engineering change starts as a patch that would ideally be small enough to upstream. We treat IOTA Foundation's daily commits as free R&D and inherit their audits, e2e tests, and fuzzing by staying close.
2. **Minimal core edits.** Consensus (Starfish), execution, networking, and storage are **never modified**. Our changes are confined to three places: (a) genesis configuration, (b) network/chain branding config, (c) a ledger of approved core diffs (target: < 20 files, each with a written justification). Everything else is built *on top of* the chain, not inside it.
3. **`qal_*` packages only.** All new smart-contract logic lives in Move packages prefixed `qal_` (`qal_framework`, `qal_credit`, `qal_data`, `qal_compute`, `qal_reserve`, `qalx`, `qal_id`, `qal_pass`), published as upgradable packages owned by multisigs. Zero logic is added to `iota-framework` itself.
4. **Test everything, inherit upstream tests.** Upstream's test suites stay green at every rebase — a red upstream suite means *our* fork broke, and we fix before proceeding. Every `qal_*` module ships with Move unit tests + happy/sad-path entry-point tests in the same PR. Property tests and fuzzing for anything that moves value.
5. **Branding at the edges.** See §3.3 — we do not mass-rename upstream crates; branding happens at config, CLI-wrapper, and UI layers.

6. **Legal gates are engineering gates.** "Phase 1 = closed-loop prepaid credit, no public transferability" is enforced in contract code (transfer restrictions on B-QALS) and in UI geo/KYC gating hooks — not just in policy documents. No phase ships past a compliance gate that hasn't been cleared with the lawyers (06 doc track).
7. **Real value follows audits.** Testnet tokens are free. The moment a contract holds anything redeemable for AUD, it has passed external audit.

---

2. Environment & Prerequisites

2.1 Hardware — The Fleet

Machine	Tailscale IP	Role in roadmap
superlocal (96 GB RAM, AMD 8060S iGPU, 32 GB VRAM)	100.73.134.20	Primary dev box, localnet host, Validator #1, dev faucet, indexer/Postgres, first compute provider (ollama GLM/Qwen)
qalcachyminirig	100.121.212.116	Validator #2 + GPU compute provider (minirig GPUs)
bb-mini	100.68.213.66	Validator #3 — low-power mini PC; Starfish's lag-tolerance is designed for exactly this
cachyos-x8664	100.111.199.12	Spare/sync node (frequently offline — do not depend on it for quorum)
Cloud VM ×1–3 (Hetzner/ AWS sydney)	public	Validator #4, public RPC gateway, explorer, faucet; geographic spread

Committee math: 4 validators tolerate 1 Byzantine fault (3f+1 = 4). Plan the grow path 4 → 7 (tolerates 2) in Phase 5/6 with external validators. Consensus traffic rides Tailscale (private, NAT-free mesh we already run); only the cloud gateway is exposed publicly.

2.2 Toolchain

- **Rust:** install via `rustup`; pin to the version in upstream's `rust-toolchain.toml` — do not freestyle. First full build of the workspace is long; budget an evening and use `cargo build --profile dev` scope-limited where possible.

- **CLI:** build the `iota` binary from the fork (`cargo install --path crates/iota-cli`) — it carries `keytool`, `client`, `move`, `validator`, `name` subcommands and is the daily driver.
- **Move:** no separate install needed — the compiler/verifier are vendored in the workspace (`external-crates/move/`). Learn via `iota move build` / `iota move test`.
- **Docker:** `docker/` in the repo ships Dockerfiles for `iota-node`, `iota-localnet`, `iota-indexer`, `iota-graphql-rpc`, `iota-faucet`, `iota-data-ingestion`, `iota-proxy`, `iota-rest-kv`, `iota-tools`. Docker + compose is the deployment format for every fleet node.
- **TS/JS:** Node 20+ (or bun) for `@iota/iota-sdk`, `@iota/dapp-kit`, `@iota/create-dapp` scaffolder, explorer UI.
- **Postgres:** required by `iota-indexer` and GraphQL — runs on superlocal.
- **Monitoring:** Prometheus + Grafana (nodes export metrics natively), alerting routed to Signal/WhatsApp via the Qalarc Hub (`signal-send`).

2.3 Repo layout (fork conventions)

```
qalnet/                    # fork of iotaledger/iota; origin = our private GitHub, upstream
  qal/                     # OUR world (everything we own lives here)
    move-packages/         # qal_framework, qal_credit, qal_data, qal_compute, qal_reserve,
    infra/                 # docker-compose per node, genesis configs, ansible-ish bootstra
    qalctl/                # our CLI wrapper
    apps/                  # dApp-kit frontends (explorer config, Qalx UI, PoR dashboard)
    docs/                  # runbooks, DR drills, decision memos
```

2.4 Skills ramp (weeks 0–6, using IOTA's own materials)

Week	Material	Deliverable proving skill
1	docs.iota.org getting-started; `docs/content/developer/getting-started/local-network.mdx`; run `iota-localnet`	Localnet up, faucet funded, first transfer via CLI
2	IOTA Move tutorials + Sui Move Book (the dialect is Sui-lineage); `iota move new/test`	Tiny coin package published to localnet with unit tests green

Week	Material	Deliverable proving skill
3	Shared objects, `Coin<T>` / `Balance<T>`, capabilities (`TreasuryCap`, `AdminCap`), upgradable packages	`qal_data::DataAnchor` spec + tests (§3.2)
4	TS SDK + dApp Kit: `createNetworkConfig`, `IotaClientProvider`, `WalletProvider`, `ConnectButton`; `@iota/create-dapp`	"Anchor a file" demo page in an app
5–6	IOTA workshops / examples repos; gas station (sponsored tx) patterns; identity.rs walkthrough	Sponsored-tx demo (Qal Pass precursor)

Rule: AI agents (qalcode/GLM) do the legwork — drafting Move, writing tests, composing PTBs — founder reviews every value-touching module. Learning is deliverable-driven, never tutorial-driven beyond week 2.

---

3. Phase 0 — Localnet Lab (Weeks 1–4)

Goal: zero infrastructure, maximum learning. Everything runs on superlocal against throwaway or 2-committee local networks. Exit with: a persistent multi-validator localnet, our first Move package tested and published, and the fork/rename strategy locked.

3.1 Localnet experiments (exact commands)

Ephemeral single-node network with a throwaway auto-funded wallet — the "hello world":

```
iota-localnet start --force-regenesis --with-faucet
```

Persistent, real multi-validator private network — the shape of Qalnet itself (2-validator committee, 60-second epochs so epoch transitions are observable in minutes, faucet on):

```
iota-localnet start --network.config persisted-localnet --with-faucet \
  --committee-size 2 --epoch-duration-ms 60000
```

Exercises to run against it (each documented in `qal/docs/`):

1. Wallet + keytool basics: create addresses, request faucet gas, transfer QALS.
2. Publish a package; call its entry functions via CLI and via a TS SDK script.
3. Object lifecycle: create a shared object, mutate it, delete it and observe the storage-deposit refund — this is the economic primitive behind data anchoring and ephemeral job payloads.
4. Epoch transitions: watch validator rotation at each 60s boundary; kill one validator mid-epoch and confirm liveness (Starfish catch-up), restart and confirm re-join.
5. `iota-swarm` in-memory networks for CI-style tests of our tooling.
6. Gas station prototype: sponsor a transaction from a second wallet (Qal Pass proof-of-concept).

3.2 First Move package — `qal_data`

The first real package is `qal_data::DataAnchor` (smallest unit of the data-layer thesis, immediately useful to the agent hub):

```
public struct DataAnchor has key, store {
    id: UID,
    owner: address,
    content_hash: vector<u8>,   // sha3-256 of payload
    uri: String,                // payload lives off-chain (MinIO on fleet / sending mach
    mime: String,
    anchored_at_ms: u64,
    signature: vector<u8>,      // producer's signature over hash
}
```

Plus a `Trail` object: append-only per-subject sequence of anchors (the Audit-Trails pattern). Deliverable: unit + integration tests, published to `persisted-localnet`, and **the first real payload anchored from the Signal/WhatsApp agent hub** by week 4.

3.3 Rename plan — iota → qal with minimal merge pain

Recommendation: do NOT mass-rename upstream crates. Renaming 104 `iota-*` crates would touch thousands of files and guarantee a conflict in nearly every upstream PR for the life of the fork — the opposite of principle #1. Instead:

Layer	Name	How
Crates/ binaries	**keep upstream names** (`iota-node` , `iota-localnet` , …)	zero diffs; upstream PRs apply cleanly
Chain/ network identity	**Qalnet**	genesis config + network config: chain ID `qalnet-*` , custom genesis.blob, our validator set — user-visible chain is ours with no source edits
User-facing CLI	**`qalctl`** (Phase 1)	thin wrapper binary embedding the `iota` CLI with Qalnet config baked in (aliases: `qalctl client` , `qalctl keytool` , …)
Coins/ tokens/docs	**QALS / doof / Qal Pass / Qal ID**	live entirely in `qal_*` Move packages, TS SDK config, and UI — never in core Rust
Optional cosmetic pass	crate renames	revisit only after 2+ clean monthly rebases have proven the workflow, and only if a milestone justifies the churn

Apache-2.0 obligations regardless: retain LICENSE and copyright notices, state significant changes in our fork's NOTICE/changelog, and don't imply IOTA endorsement of "Qalnet".

Phase 0 exit criteria: persistent 2-validator localnet stable across 100+ epochs; `qal_data` published with tests; kill-a-validator drill documented; rename memo signed off; fork repo + CI (upstream test suites) green.

(Optional parallel pilot per architecture §3 decision A→B: deploy a QALS coin on public IOTA mainnet as a live tokenomics sandbox. Cheap, reversible, non-blocking — run it only if it doesn't distract from the localnet lab.)

---

4. Phase 1 — Devnet on The Fleet (Months 2–3)

Goal: `qalnet-dev-1`: a real 4-validator network across our own machines, with monitoring, explorer, faucet, and our CLI. This is the moment the fork becomes an operations project.

4.1 Genesis

Build a custom `genesis.blob` with `iota-genesis-builder` (+ `iota-genesis-common`):

- **Chain:** `qalnet-dev-1`. Native coin QALS, 6 decimals (1 Qals = 1,000,000 doofs) (smallest unit = **doof**). Total supply **4,600,000,000 QALS**, minted at genesis, no protocol inflation (validator rewards come from the dedicated 5% subsidy allocation, not % inflation).
- **Allocations** (architecture §5.2): ecosystem/rewards 25% (1.15B), treasury 20% (0.92B), team 15%, investors 10%, compute rewards 10% (0.46B), Qalx liquidity 10%, validator subsidy 5% (0.23B), identity-gated user airdrop 5%. Treasury/team/investor buckets land under multisig custody with vesting enforced by Move (not by trust).
- **Validators:** 4 — superlocal, qalcachyminirig, bb-mini, cloud VM. Validator identity/protocol keys generated per node, account keys in Stronghold-backed stores, all backups tested before the network's first epoch.
- **Parameters:** 24h epochs (devnet may run shorter, e.g. 6h, to exercise rotation); gas: fixed low QALS price, 50% burned / 50% to validators; refundable storage deposits; committee changes via config, not code.

4.2 Infrastructure

- **Docker-compose per node** from the repo's `docker/` images (`iota-node` everywhere; `iota-faucet` on superlocal; `iota-indexer` + Postgres + `iota-graphql-rpc` on superlocal or cloud).
- **Networking:** p2p consensus over Tailscale; public full-node/RPC gateway on the cloud VM for apps and external tools.
- **Monitoring:** Prometheus scraping node metrics → Grafana dashboards (consensus latency, tx rate, epoch countdown, disk); alerts → Signal via the hub. Uptime checks per validator.
- **Explorer:** fork the upstream Sui-lineage explorer UI, wired to our indexer/GraphQL; rebrand Qalnet. Public at Phase 5 (Doofnet), internal until then.

- `qalctl` **v1:** wraps the `iota` CLI (client, keytool, validator ops) with Qalnet endpoints/config baked in; subcommands for faucet, publish, epoch status.

4.3 Ops discipline from day one

- DR drill #1: restore a validator from bare metal + key backup in < 2h. Documented.
- Rebase drill #1: rebase fork onto upstream `main`, run full upstream test suite + our e2e smoke, redeploy devnet with no state loss expectations (devnet is disposable; the *process* is what we're testing).
- Everything as code: node configs, genesis inputs, compose files in `qal/infra/`.

Exit criteria: 4-validator devnet runs 30 days with > 99.5% uptime; one unannounced validator kill + restore survived; explorer shows live data; faucet serves CLI/scripted requests; all 4.6B QALS accounted for on-chain in multisig/vesting objects.

---

5. Phase 2 — Core Contracts (Months 3–5)

Goal: the economic heart in audited Move: credit accounts, data anchoring in production use, and the AU$1-backed reserve.

5.1 Package list

Package	Contents
`qal_framework`	QALS utilities, admin capabilities, upgrade governance glue, shared errors/events
`qal_credit`	**CreditAccount** (per-user/app prepaid balance), **Hold** (pre-authorization escrow — the primitive every marketplace draws on), **Subscription** (recurring metered draws from an account)
`qal_data`	DataAnchor + Trails (hardened from Phase 0), per-app trails, producer signature verification
`qal_reserve`	B-QALS / G-QALS coins, Reserve object, FloorVault with enforced ratchet
`qal_pass`	Gas sponsorship rules: who/what the treasury sponsors, per-app budgets, rate limits (Gas Station pattern)

5.2 The Reserve (per `QALS_BACKING_DESIGN.md`) — the invariant as code

Two `Coin<T>` types (`b_qals::B_QALS`, `qals::QALS`) + one shared `Reserve` object. Structural guarantees, not bookkeeping:

- **Mint gate:** `mint_against_aud()` callable only by the Oracle Committee after a confirmed AUD deposit into the segregated bank/PSP account; rate-limited (≤ AU$250k/day in this phase).
- **Redemption desk:** 1 B-QALS → AU$1 − 0.5% fee; 24h time-lock per account for redemptions > AU$10k (fraud/AML window). Full reserve = no bank-run insolvency risk.
- **Consumption burn:** service usage burns B-QALS and moves its AU$ from Reserve to operating revenue — "1 QALS = AU$1 of qalarc credit" is true by construction.
- **FloorVault:** funded by 20% of service gross margin + 100% of redemption fees + 50% of Qalx fees; floor ratchet enforced by a `Ratchet` typestate (can only rise, governance-raised with 48h timelock). G-QALS floats above it; parity graduation to full AU$1 backing is an explicit future event, not a promise.
- **Invariant:** `AUD_in_reserve ≥ B_QALS_outstanding × AU$1.00` at every block, provable monthly. Reconciliation delta > 0.5% auto-pauses mint.

Governance/keys: Oracle Committee = **3-of-5 multisig** (superlocal, minirig, bb-mini, cloud signer, one external key) publishing signed reserve statements on-chain; Treasury Council = **2-of-3 multisig + 48h timelock** for parameter changes; emergency pause auto-expires in 72h with mandatory public post-mortem.

Proof-of-Reserves dashboard (ships with the package, not after): wiki page + API showing live B-QALS supply (the chain *is* the liability ledger), attested AUD balance, coverage ratio, history. Monthly: bank statement + reconciliation letter; quarterly: accountant AUP check.

5.3 Transfer restriction (legal gate as code)

B-QALS transferability is restricted at the contract level in this phase (internal credit ledger per the closed-loop prepaid framing). The unlock is a deliberate, lawyered event tied to VASP registration (Phase 5/6), implemented as a transfer-policy object the council can flip — visible, auditable, reversible to safe state.

Exit criteria: packages published to devnet with full test suites; CreditAccount paying a real qalarc service (AI proxy billing) end-to-end; hub payloads anchoring via `qal_data` daily;

Reserve demo cycle (deposit → mint → consume-burn → redeem) exercised on devnet; Qal Pass sponsoring app transactions; internal Move review complete.

---

6. Phase 3 — Compute Marketplace v1 on Real Jobs (Months 5–8)

Goal: GPU-hours become escrowed, metered, provable objects — starting with our own workloads so the marketplace never has a cold-start problem (we are the anchor tenant).

6.1 `qal_compute` package

- `Provider` (accelerator class, benchmark score, price, reputation stake, active flag) — registration gated by a Hierarchies accreditation (`device.can-run-jobs`) so a compromised rig is revocable without code changes.
- `ComputeJob` lifecycle: `POSTED → MATCHED (escrow locked from a Hold) → RUNNING → ATTESTED → SETTLED / SLASHED / REFUNDED`. Move's linear types make the escrow un-leakable; settlement releases `min(actual, escrow)` with the remainder refunded.
- `Attestation`: provider DID signature over `(job_id, outputs_hash, gpu_seconds)`, output + telemetry DataAnchors, optional verifier countersign.

6.2 v1 scope — the hub LLM proxy first

The first real workload is **LLM inference metered per token** through the qalarc hub (GLM/Qwen served by ollama on superlocal's 8060S — already running today). Sequence: v0 fake jobs on devnet proving escrow/settle/slash in Move tests (2 wks) → v1 real proxy traffic (≈6 wks): every proxied request creates/extends a job, settlement debits the requester's CreditAccount, provider (us) gets paid, a **ComputeReceipt NFT** is minted (job ID, model+input hashes, GPU-seconds, cost, provider DID) — simultaneously our billing record and the AI-output provenance backbone for endispute/doof. **PriceTable** object quotes doofs per GPU-second per accelerator class; agents compare providers by `price × benchmark-normalised-time` from the **Qal Bench** oracle (weekly fleet benchmark runs, medianised on-chain).

6.3 v1.5 (still inside the phase)

- minirig + bb-mini as additional providers (multi-provider matching, reputation-weighted queue).
- `JobBundle` batching so per-inference micro-jobs amortise to one sponsored settlement.

- Optimistic verification for high-value jobs (second node re-runs sampled inputs; mismatch → dispute → slash).

Exit criteria: ≥ 30 consecutive days of real proxy jobs settled on-chain with zero accounting drift between CreditAccounts, escrow objects and hub billing; first ComputeReceipt NFTs minted; slash path exercised deliberately in a fault-injection test; unit economics spreadsheet reconciled against on-chain actuals.

---

7. Phase 4 — Qalx v0–v1 (Months 6–10, overlaps Phase 3)

Goal: the financial layer goes interactive — redemption/trust surfaces first, AMM second, UI everywhere.

- **Qalx v0 (parity desk — target land early, ~month 6):** Reserve redemption desk UI + Reserve mirror + Proof-of-Reserves page. Per `QALX_DEFI_PLATFORM.md` build order: trust comes first; everything else can wait. (The contracts landed in Phase 2; this is the product surface.)
- **Qalx v1 (AMM, ~months 7–9):** `qalx::pool` constant-product AMM — `create_pool`, `swap_exact_*` with `min_out` + deadline, LP positions as native `Coin<LP>` objects. Fee 25 bps (20 LPs / 5 treasury). Launch pool: **G-QALS/B-QALS** (utility price discovery), treasury-seeded from the 10% liquidity allocation. **FloorBot**: always-on bid at `floor − ε` backed by FloorVault — the floor as a visible order, not a promise. Parity rail (B-QALS⇄qAUD) activates with qAUD in Phase 6.
- **UI (continuous):** React + `@iota/dapp-kit` + `@iota/iota-sdk` (`createNetworkConfig`, `IotaClientProvider`, `WalletProvider`, `ConnectButton`; scaffolded with `@iota/create-dapp`) in `qal/apps/`: swap, pools, positions; PTBs composed client-side with slippage guards. "Connect Qal ID" becomes the standard sign-in across qalarc apps as identity lands (Phase 5).
- **No public mempool sandwiching; flash-loan resistance** by reading Reserve state (not pool price) for mint/redeem truth.

Exit criteria: FloorBot sustained ≥ 30 days without intervention; AMM fee accounting verified against the 20/5 split in tests + mainnet-state assertions; UI live internally; Qalbook CLOB evaluation memo (see Phase 6) drafted with port-cost numbers.

---

8. Phase 5 — Identity, Doofnet, Audit, Compliance (Months 9–12)

Goal: the private chain meets the outside world — deliberately, in this order: identity → public testnet → external audit → compliance file.

1. **Qal ID (identity rollout).** Reuse/fork `identity.rs` (Apache-2.0, active) for W3C DIDs/VCs anchored on Qalnet (`qal_id` package: DID anchoring + status lists). Humans *and* AI agents get DIDs; keys in Stronghold wrapped by OS enclaves. **Agent credentials** via the Hierarchies pattern: root authority = qalarc issuer; accreditations like `agent.hub.can-send-signal`, `agent.spend-cap: 50 QALS/day`, `device.minirig.can-run-jobs` — validated off-chain in µs via WASM, enforced on-chain wherever money moves. Hub agents get DID wallets with chain-enforced per-epoch spend caps and revocation drills.
2. **Doofnet — public testnet.** Public faucet (rate-limited, identity-gated), public explorer, public RPC, the airdrop allocation (5%) rehearsed as an identity-gated drop — our identity pilot wearing a party hat. Geo/AML gating hooks ship in all UIs now, not later.
3. **Audit #1.** External Move audit of all `qal_*` packages (~US$30–80k, per the Qalx doc's disclosure ladder) *before* mainnet-with-real-value. Findings fixed, re-reviewed, regression-tested in CI. Bug-bounty program (QALS-denominated) scoped.
4. **AUSTRAC VASP preparation (parallel with lawyers per `06_bank_exchange`).** DCE registration path, AML/CTF program draft, KYC tiering (tier 0: AU$500 / tier 1: AU$10k / tier 2: AU$100k+ via partner KYC), transaction-monitoring design. Engineering deliverables: KYC-tier objects linked to Qal ID, tier-gated limits enforced in Reserve/Qalx entry points, reporting data exports. This track runs concurrently from month 9; it gates Phase 6, not Phase 5.
5. **Devnet → mainnet-internal readiness:** validator DR drill #2 (full site-loss simulation on one fleet machine), key ceremony rehearsal for the multisigs, runbook hardening.

Exit criteria: Doofnet public for 60 days with external test users; identity-gated airdrop executed end-to-end; audit #1 passed with all high/critical findings closed; AUSTRAC file submitted or submission-ready (lawyer-confirmed); agent revocation drill passed.

---

9. Phase 6 — Year 2: External Surfaces

Sequential, each gated on the previous + legal sign-off:

1. **External venue.** Registered spot venue for QALS/qAUD pairs post-VASP (KYC-tiered, same safety stack). Qalx opens past the wall.
2. **qAUD.** Tokenized AUD claim under the licensed/registered arrangement: 1 qAUD → AU$1, safeguarded deposits, Reserve tranche in AU T-Bills (> AU$250k balance) with yield to FloorVault. Parity pool B-QALS/qUSD⇄qAUD goes live; redemption arbitrage pins it.
3. **Receivables vault (Qalx v2).** ComputeReceipt NFTs as collateral — lenders supply qAUD/B-QALS, borrowers draw 60–80% LTV; default = claim the on-chain escrow (no oracle guesswork). Endispute dispute-settlement invoices join the same vault pattern.
4. **Qalbook CLOB evaluation → execution.** DeepBook v3 fork (Apache-2.0 verified): keep Book/State/Vault + BalanceManager + `swap_exact_*` + flash loans; strip DEEP tokenomics (→ QALS-denominated fees), referral, margin, predict. Trigger: utility-pool volume > AU$1M/mo or external demand post-VASP. Budget honestly: 2–3 Move engineers × ~2 months (sui→iota framework reconciliation across ~15k lines) **plus** external audit ~US$150–400k before it touches real money. If the numbers don't clear, the AMM + RFQ intents hybrid continues to serve.
5. **Validator expansion** 4 → 7 (external validators admitted with stake + accreditation), anchoring service maturation (Qalnet checkpoint roots → public IOTA mainnet on every checkpoint), G-QALS floor-coverage dashboard as the public honesty metric.

---

10. Workstream Map

Workstream	Phases active	Owner type	Exit criteria (final)
Node/fork engineering (rebase, genesis, config)	0–6 continuous	Founder + AI agents	Monthly rebase cadence ≤ 48h turnaround, 6 consecutive clean cycles
Move contracts (`qal_*`)	0–5	AI-agent-drafted, founder-reviewed, externally audited	All packages audited, tests green, upgrade path exercised

Workstream	Phases active	Owner type	Exit criteria (final)
Infra/ops (Fleet, monitoring, DR)	1–6	Ops automation + AI agents	99.5% uptime; 2 DR drills passed; alerts → Signal live
Compute marketplace	3–4	AI agents (bots/oracle) + founder	30 days real jobs, zero accounting drift
DeFi (Qalx → Qalbook)	2, 4, 6	Founder + AI agents + audit	v1 live; CLOB go/no-go memo decided on numbers
Identity (Qal ID, agent creds)	5 (design from 2)	Founder + AI agents	Airdrop executed; revocation drill passed
UI/apps (dApp-kit)	3–6	AI-agent-led, founder review	PoR dashboard, Qalx UI, explorer public
Compliance/legal	2–6 (parallel)	External lawyers + founder	VASP/DCE cleared before external transferability
Security/audit	5–6	External firms	Audit #1 passed pre-real-value; bounty live
Docs/runbooks/ branding	all	AI agents	Every drill + runbook in `qal/docs/`; brand at edges only

---

11. Definition of Done — per phase

- **Phase 0:** persistent localnet 100+ epochs stable · `qal_data` published + tested · validator-kill drill documented · rename memo approved.
- **Phase 1:** 4-validator devnet 30 days > 99.5% uptime · restore-from-backup < 2h · explorer/faucet/qalctl live · 4.6B QALS fully accounted in custody objects.
- **Phase 2:** CreditAccount bills a real service · Reserve mint/consume/redeem cycle proven · PoR dashboard live · B-QALS transfer restriction verified by negative tests.
- **Phase 3:** 30 days real GPU jobs settled · receipts minted · slash path fault-injection-tested · billing reconciles to the doof.

- **Phase 4:** FloorBot 30 days autonomous · AMM fee split verified · internal UI live · CLOB memo written.
- **Phase 5:** Doofnet 60 days public · airdrop identity-gated end-to-end · **audit #1 passed** · AUSTRAC file submitted.
- **Phase 6:** each surface ships only behind its legal gate; every multisig change through timelock; post-launch 30-day stability review per launch.

---

12. Key Technical Risks & Mitigations

Risk	Severity	Mitigation
Port risk (DeepBook→Qalbook; sui→iota framework drift is consensus-critical)	High	Defer to Phase 6 behind a volume trigger; budget 2–3 Move engineers × 2 months + $150–400k audit; AMM+RFQ hybrid is the standing fallback
Upstream drift (iotaledger/iota commits daily)	Medium	**Monthly rebase cadence**, fixed ops window: fetch upstream → merge → full upstream + our test suites → staging deploy → fleet rollout. Core-edit ledger stays < 20 files; any new core diff requires a written justification + rebase-impact note. If IOTA stalls, track MystenLabs/sui directly — our investment is portable (Apache-2.0, Sui-lineage)
Security bug in inherited or new code	High	Inherit upstream audits/tests; keep them green; **external audit before real value moves** (audit #1 gates Phase 5→6); bug bounty; key ceremonies + Stronghold; agent keys capped by chain-enforced spend limits; emergency pause + post-mortem duty
Validator outage / residential internet (home machines in committee)	Medium	Starfish lag-tolerance by design; cloud validator keeps quorum; Tailscale mesh; DR drills (2× before mainnet-internal); Prometheus→Signal alerts; cachyos-x8664 kept out of quorum math
Regulatory timing (token classification, VASP delay)	High	Closed-loop prepaid framing enforced in code; every external-facing feature sits behind a legal gate; lawyers engaged from Phase 2, continuously (06 doc)

Risk	Severity	Mitigation
Skills ramp (Move/Rust depth on a small team)	Medium	Deliverable-driven ramp (§2.4) using IOTA's own workshops/docs; AI agents do legwork; EVM compatibility deliberately deferred to shrink the surface; upstream test suites as guardrails
Shared-object contention (single Pool serializes trades)	Low-Med	Monitor from day one; sharded pools are the known escape hatch; internal volumes are modest at this scale

---

13. What We Deliberately Do NOT Build

1. **No perps/derivatives early.** Financial products under the Corporations Act → AFS-licence territory. Revisit only after the venue is registered and volumes justify the compliance cost (Qalx v3+, optional).
2. **No public sale or listing until licensed.** QALS is prepaid service credit with contract-enforced transfer restrictions until the AUSTRAC/lawyer gates clear. No exceptions, no "soft launches".
3. **No EVM day one.** The component map confirms the iota node repo contains **no EVM implementation** — EVM on the fork means a separate stack (wasp-style emulation or a ported adapter). Revisit Phase 4+ as a compatibility layer for partners; value-creation logic stays in Move regardless ("Move for value, EVM for shims" — if at all).
4. **No liquid staking, no bridged external assets, no public validators** before Phase 5/6 — each adds attack surface and regulatory texture before the core loops (credit → consume → burn; job → escrow → settle) are proven.
5. **No custom consensus or core-protocol modifications.** Starfish, execution, and storage ship as-is. Our differentiators are the `qal_*` object economy and ops excellence — not a bespoke BFT.
6. **No ternary revival.** The trit is a merch joke. Binary everywhere.

Companion docs: `03_qals_architecture/` *(architecture, backing),* `04_credit_token_platform/CREDIT_PLATFORM.md`, `05_defi_trading/` *(Qalx, DeepBook review),* `06_bank_exchange/BANK_EXCHANGE_PLAN.md` *(compliance owner),*

`07_compute_marketplace/COMPUTE_MARKETPLACE.md`, `08_nft_strategy/` *(Doofs, ComputeReceipts)*, `09_identity_ai/IDENTITY_AND_AI.md`.